

Making the Transition to Linux

A Guide to the Linux Command Line Interface for Students

Joshua Glatt

Making the Transition to Linux: A Guide to the Linux Command Line Interface for Students

Joshua Glatt

jzglatt@wustl.edu

Copyright © 2008 Joshua Glatt

| Revision History | | |
|---|--------------|-----|
| Revision 1.31 | 14 Sept 2008 | jpg |
| Various small but useful changes, preparing to revise section on vi | | |
| Revision 1.30 | 10 Sept 2008 | jpg |
| Revised further reading and suggestions, other revisions | | |
| Revision 1.20 | 27 Aug 2008 | jpg |
| Revised first chapter, other revisions | | |
| Revision 1.10 | 20 Aug 2008 | jpg |
| First major revision | | |
| Revision 1.00 | 11 Aug 2008 | jpg |
| First official release (w00t) | | |
| Revision 0.95 | 06 Aug 2008 | jpg |
| Second beta release | | |
| Revision 0.90 | 01 Aug 2008 | jpg |
| First beta release | | |

License

This document is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States License [<http://creativecommons.org/licenses/by-nc-sa/3.0/us/>].

Legal Notice

This document is distributed in the hope that it will be useful, but it is provided “as is” without express or implied warranty of any kind; without even the implied warranties of merchantability or fitness for a particular purpose.

Although the author makes every effort to make this document as complete and as accurate as possible, the author assumes no responsibility for errors or omissions, nor does the author assume any liability whatsoever for incidental or consequential damages in connection with or arising out of the use of the information contained in this document.

The author provides links to external websites for informational purposes only and is not responsible for the content of those websites. The inclusion of a link to an external website from anywhere in this document does not imply an endorsement of that site nor of the information, products, or services offered there.

Trademarks

Linux is a registered trademark of Linus Torvalds. Microsoft and Windows are registered trademarks of Microsoft Corporation. Mac OS is a registered trademark of Apple Inc. UNIX is a registered trademark of The Open Group. Google is a trademark of Google Inc. Wikipedia is a registered trademark of the Wikimedia Foundation, Inc. Eclipse is a trademark of the Eclipse Foundation, Inc. GNOME is a trademark of the GNOME Foundation. KDE and K Desktop Environment are trademarks of KDE e.V. O'Reilly is a registered trademark of O'Reilly Media, Inc. All other trademarks and registered trademarks are the property of their respective owners. Use of a term in this document should not be regarded as affecting the validity of any trademark or service mark.

Table of Contents

| | |
|--|------|
| Introduction | vii |
| 1. What is this guide for? | vii |
| 2. Who would want to read this guide? | vii |
| 3. Who would not want to read this guide? | vii |
| 4. What is the reader expected to know? | vii |
| 5. What does the reader need? | viii |
| 6. What does this guide cover? | viii |
| 7. What does this guide not cover? | viii |
| 8. Other information about this guide | viii |
| 9. Conventions used in this guide | ix |
| 10. Structure of this guide | ix |
| 11. Suggestions for using this guide | x |
| 1. Getting started with Linux | 1 |
| 1.1. User interfaces in Linux: graphical and command line | 1 |
| 1.2. Graphical user interfaces (GUIs) for Linux | 1 |
| 1.3. Accessing the Linux CLI through a GUI's terminal window | 2 |
| 1.4. Shells, the shell prompt, and your home directory | 2 |
| 1.5. Key differences between Windows/Mac OS X and Linux | 3 |
| 1.6. Understanding the tables of commands and options | 4 |
| 2. Built-in help system | 5 |
| 2.1. Manual ("man") pages | 5 |
| 2.2. GNU info pages | 5 |
| 3. Basic shell commands and related utilities | 6 |
| 3.1. Features of the shell | 6 |
| 3.2. Navigating the file system | 7 |
| 3.3. Manipulating the file system | 9 |
| 3.4. Handling compressed files (tarballs) | 11 |
| 3.5. Working with text streams | 11 |
| 3.6. Using job control (partial coverage) | 13 |
| 4. SSH (secure) file transfer (SFTP) | 15 |
| 5. Text editors | 18 |
| 5.1. vi | 18 |
| 5.2. GNU Emacs | 18 |
| 6. Programming tools | 23 |
| 6.1. Compiling programs | 23 |
| 6.2. Running programs | 24 |
| 6.3. Debugging programs | 25 |
| 6.4. Inspecting and modifying programs | 27 |
| A. Further reading | 29 |
| A.1. Suggested resources for finding information | 29 |
| A.2. More on Linux | 29 |
| A.2.1. General Linux resources | 29 |
| A.2.2. Linux file system | 30 |
| A.3. More on the shell and related utilities | 30 |
| A.3.1. General information on the Bash shell | 30 |
| A.3.2. Changing your shell temporarily to Bash | 31 |

| | |
|--|----|
| A.3.3. Text processing with the shell | 31 |
| A.3.4. Shell scripting (using Bash) | 32 |
| A.3.5. The GNU Core Utilities (coreutils) | 32 |
| A.4. More on text editors | 32 |
| A.4.1. More on vi | 32 |
| A.4.2. More on GNU Emacs | 33 |
| A.5. More on programming on Linux machines | 33 |
| A.5.1. Linux programming and the GNU toolchain | 33 |
| A.5.2. More on GCC | 34 |
| A.5.3. GNU Make and Makefiles | 34 |
| A.5.4. More on GDB | 34 |
| A.5.5. More on the GNU Binary Utilities (binutils) | 35 |
| B. Graphical alternatives | 36 |
| B.1. GUI-based file managers | 36 |
| B.2. GUI-based file transfer programs | 36 |
| B.3. GUI-based text editors | 36 |
| B.3.1. GUI-based vi and Emacs | 36 |
| B.3.2. Other GUI-based text editors | 36 |
| B.4. GUI-based programming and debugging tools | 37 |
| B.4.1. Integrated Development Environments (IDEs) | 37 |
| B.4.2. GUI-based debuggers not part of an IDE | 37 |

List of Tables

| | |
|---|----|
| 2.1. Commands for the built-in help system | 5 |
| 3.1. Commands for navigating the file system | 8 |
| 3.2. Common symbols from the file system | 9 |
| 3.3. Commands for manipulating the file system | 10 |
| 3.4. Commands for handling compressed files (tarballs) | 11 |
| 3.5. Standard streams | 12 |
| 3.6. I/O redirection operators | 13 |
| 3.7. Commands for working with text streams | 13 |
| 3.8. Commands for terminating a process | 14 |
| 4.1. SSH file transfer (SFTP) commands | 17 |
| 5.1. Basic Emacs commands | 20 |
| 5.2. Moving the cursor in Emacs | 21 |
| 5.3. Manipulating text in Emacs | 22 |
| 5.4. Additional Emacs commands | 22 |
| 6.1. Commonly used GCC flags | 24 |
| 6.2. Commonly used GDB commands, Part 1 | 26 |
| 6.3. Commonly used GDB commands, Part 2 | 27 |
| 6.4. Selected commands from GNU Binary Utilities (binutils) | 28 |

List of Examples

| | |
|---|----|
| 1.1. How to read a table entry | 4 |
| 3.1. Using cd | 7 |
| 3.2. Using du | 8 |
| 3.3. Using find | 8 |
| 3.4. Using ls | 8 |
| 3.5. Using ln | 9 |
| 3.6. Using mv | 10 |
| 3.7. Using rm | 10 |
| 3.8. Using cat with pipes | 12 |
| 3.9. Using /dev/null | 12 |
| 3.10. Using diff | 12 |
| 3.11. Using echo | 12 |
| 3.12. Using ps and kill | 14 |
| 4.1. Using SSH file transfer (SFTP) | 16 |
| 6.1. Using GCC | 24 |
| 6.2. Using objdump | 27 |

Introduction

1. What is this guide for?

This guide is intended to introduce computer users, particularly students, to the command line interface (CLI) and the command line programming tools of the Linux operating system.

2. Who would want to read this guide?

Students taking Linux-based programming courses (particularly C/C++-related courses) who have little or no previous experience with Linux would find the information in this guide to be helpful in working on the assignments for those courses.

Students taking other Linux-based courses or doing work that requires the use of the Linux command line interface (CLI) may find this guide to be helpful in working on Linux-based projects.

Students or others who access a Linux server via secure shell (SSH)² may find parts of this guide to be helpful in working with Linux over SSH.

Students or others who are interested in learning how to use Linux may find this guide to be helpful in becoming comfortable with the CLI and in locating resources for enhancing their knowledge of Linux.

3. Who would not want to read this guide?

Those looking only for information on programming (such as how to program in language X) or on computer science would not find this guide to be helpful.

Those looking for in-depth information on the Linux desktop environments would not find this guide to be helpful, although some information about applications for Linux desktop environments is provided in Appendix B, *Graphical alternatives*.

Those looking for information on installing Linux or on administering their own Linux installation would not find this guide to be helpful **except** perhaps as a supplemental resource for learning how to use Linux, as described in these suggestions.

4. What is the reader expected to know?

The reader is assumed to be proficient in general computer use with an operating system whose primary means of interaction with the user is through a graphical user interface (GUI), such as Microsoft® Windows® or Mac OS® X. No previous experience with UNIX® or Linux is assumed.¹

¹A historical note: as described in Randal Bryant and David O'Hallaron's *Computer Systems: A Programmer's Perspective* (Upper Saddle River: Prentice Hall, 2003), the Unix operating system was first developed at Bell Laboratories from 1969 through 1974

For the discussion in Chapter 6, *Programming tools*, the reader is assumed to be familiar with programming and with standard programming tools, such as compilers and debuggers.

5. What does the reader need?

To get any real benefit from the guide, the reader must have access (whether physical access or via SSH)² to a computer running Linux. Similarly, for the reader to benefit from sections in the guide that cover specific programs (such as Section 5.2, “GNU Emacs”), the software under discussion must be installed.

6. What does this guide cover?

This guide covers topics related to the use of the Linux command line interface (CLI) and its programming and debugging tools, including the man and info pages, the shell, SSH (secure) file transfer, text editors, and the command line tools needed to compile, run, debug, and inspect programs (primarily the GNU Compiler Collection [GCC] and the GNU Debugger [GDB]). The guide also includes references to further reading on many of these topics.

7. What does this guide not cover?

This guide does not cover any material related to computer science or programming. Similarly, this guide does not explain what compilers, assemblers, linkers, or debuggers are, nor does it explain how to use those tools.

This guide does not cover installing Linux, nor does it cover Linux system administration.

This guide does not cover the use of applications available exclusively through a Linux desktop environment, although a listing of graphical alternatives to the command line tools is provided in Appendix B, *Graphical alternatives*.

8. Other information about this guide



Note

Feedback and errata reports are greatly appreciated and can be sent by e-mail to me (the author) [mailto:jzglatt@wustl.edu].

The most recent version of this guide is available at <http://students.cec.wustl.edu/~jg18/guide/>.

(14). The GNU Project³ (GNU is short for GNU's Not Unix) was established in 1984 with the "goal of developing a complete Unix-like system whose source code is unencumbered by restrictions on how it can be modified or distributed" (5) - that is, made of free software. Linux was created from the combination of GNU software with the Linux kernel, a "Unix-like operating system kernel" first developed in the early 1990s under the direction of Linus Torvalds, who at the time was a graduate student at the University of Helsinki (18).

²Wikipedia's article [http://en.wikipedia.org/wiki/Secure_Shell] on SSH describes it as "a network protocol that allows data to be exchanged using a secure channel between two networked devices." A discussion of SSH is beyond the scope of the guide, but the Wikipedia article may be helpful in learning more. If you're a student, you can try asking at the help desk at your school's computing center.

This guide was prepared as DocBook [<http://www.docbook.org/>] V4.5 XML source using Aquamacs Emacs [<http://aquamacs.org/>]'s nXML Mode [<http://www.thaiopensource.com/nxml-mode/>].

The XSLT and FO processing was done with xsltproc [<http://xmlsoft.org/XSLT/xsltproc2.html>] and Apache FOP [<http://xmlgraphics.apache.org/fop/>], respectively.

I found the book *Version Control with Subversion* [<http://svnbook.red-bean.com/>] to be helpful with managing the guide as a Subversion [<http://subversion.tigris.org/>] repository.

I found *DocBook: The Definitive Guide* [<http://www.docbook.org/tdg/en/html/docbook.html>], the DocBook Wiki [<http://wiki.docbook.org/topic>], *DocBook XSL: The Definitive Guide* [<http://www.sagehill.net/docbookxsl/>], and *DocBook XSL Stylesheets: Reference Documentation* [<http://docbook.sourceforge.net/release/xsl/current/doc/>] to be extremely helpful with preparing and publishing the DocBook source.

This guide was adapted from “A Guide to the Linux Command Line Interface for Computer Science Students at Washington University,” my final paper for EP 310 (Technical Writing) from Summer 2008 at Washington University.

The layout of this guide was inspired in part by that of the guides at The Linux Documentation Project (TLDP) [<http://www.tldp.org/guides.html>].

9. Conventions used in this guide



Note

The text in the HTML and PDF versions of the guide is not formatted in the same way (for example, text that appears in `monospace` in the HTML versions might not appear in `monospace` in the PDF version). I am still working on fixing these discrepancies as best as I can.

Commands and other user input to be typed at the prompt appear in bold, as in **ls** (for a simple command) and **./lab0** (for more complicated input).

Text that would appear on the user's screen looks like this:

```
foo: Command not found.
```

Also see Example 1.1, “How to read a table entry”.

10. Structure of this guide

This guide is organized as a collection of short introductions to selected topics related to the use of the Linux CLI and programming tools:

A. Linux and its user interfaces, discussed in Chapter 1, *Getting started with Linux*, with references for further reading in Section A.2, “More on Linux”.

- B. The built-in help system, discussed in Chapter 2, *Built-in help system*.
- C. The shell, discussed in Chapter 3, *Basic shell commands and related utilities*, with references for further reading in Section A.3, “More on the shell and related utilities”.
- D. SSH (secure) file transfer (SFTP), discussed in Chapter 4, *SSH (secure) file transfer (SFTP)*.
- E. Text editors, discussed in Chapter 5, *Text editors*, with references for further reading in Section A.4, “More on text editors”.
- F. Programming tools, discussed in Chapter 6, *Programming tools*, with references for further reading in Section A.5, “More on programming on Linux machines”.

There are also general suggestions for finding information in Section A.1, “Suggested resources for finding information”.

Finally, for your convenience, I have included references to GUI-based versions of these tools in Appendix B, *Graphical alternatives*.

11. Suggestions for using this guide



Tip

The information in Section 1.5, “Key differences between Windows/Mac OS X and Linux” and Section 3.1, “Features of the shell” can save you a lot of time and frustration when you are working with the CLI.

The resources mentioned in Section A.1, “Suggested resources for finding information” can help you find specific information quickly.

Although you *could* read this guide from start to finish, you may find it useful to read whichever sections interest you, or you might want to try a strategy such as one of the following:

- If you just want to get started immediately and would rather learn the details later:
 1. Open a terminal (check Section 1.3, “Accessing the Linux CLI through a GUI's terminal window” if necessary) if you don't have one open already.
 2. Start with Section 3.2, “Navigating the file system”.
 3. Read any other sections that interest you.
 4. Then consider trying the suggestions listed below.
- If you're taking a course or doing work that involves programming:
 1. Start with Section 1.3, “Accessing the Linux CLI through a GUI's terminal window” if you don't know how to open a terminal.

2. Then look at Section 1.4, “Shells, the shell prompt, and your home directory” if you've never worked with a shell before.
 3. Look over Section 9, “Conventions used in this guide”, including Example 1.1, “How to read a table entry”, if you haven't done so already.
 4. Then take a quick look at Chapter 2, *Built-in help system* so that you at least know about the man and info pages.
 5. Read Section 3.2, “Navigating the file system” and the sections following it to learn how to work with the shell, using the file managers discussed in Section B.1, “GUI-based file managers” if you get stuck.
 6. Choose one of the editors discussed in Chapter 5, *Text editors* and learn how to use it, resorting to one of the graphical text editors discussed in Section B.3, “GUI-based text editors” if you get stuck.
 7. Then read Section 6.1, “Compiling programs” and the two sections that follow it to learn how to compile, run, and debug your programs using Linux. I suggest that you wait to try the graphical programming tools mentioned in Section B.4, “GUI-based programming and debugging tools” until after you've had some experience with the command line tools.
 8. If you need to transfer files over a network, read Chapter 4, *SSH (secure) file transfer (SFTP)* (or Section B.2, “GUI-based file transfer programs”).
 9. After you have the basics down, check Appendix A, *Further reading* to learn more about Linux.
- If you're taking a course or doing work that doesn't involve programming:
 1. First complete the steps above for programming-related courses/work, skipping the step on programming tools.
 2. Then take a look at Appendix A, *Further reading* and Appendix B, *Graphical alternatives* for information related to your course or work.
 - If you're connecting to a Linux machine via SSH:²
 1. Clearly, the information in Appendix B, *Graphical alternatives* is irrelevant, unless you want to try X tunneling.³
 2. So take a look at the steps above for programming-related courses/work and complete them as is relevant to your objectives.
 3. If you need to transfer files between your machine and the server, you'll need a separate file transfer program, which might be SFTP (either the command line version discussed in Chapter 4, *SSH (secure) file transfer (SFTP)* or graphical versions such as those mentioned in Section B.2, “GUI-based file transfer programs”) or a different program altogether.

- If you're interested in learning how to use Linux:
 1. If you're installing Linux on your computer (which is recommended if you're serious about becoming proficient in using Linux), select a Linux distribution to use and read its documentation or books about it as needed.⁴ Be sure to carefully examine the distribution's website for useful information.

If you need help selecting a distribution, you may wish to consider Dr. Norm Matloff's opinion in his *Beginner's Guide to Installing and Using Linux* [<http://heather.cs.ucdavis.edu/~matloff/Linux/LinuxInstall.pdf>] (available only in .pdf format).

2. Once you have access to a machine running Linux, follow the steps above for programming-related courses/work to become familiar with the CLI.

You can skip the step on programming tools, although you may find that programming experience with C or C++ is helpful in understanding Linux.

3. Then carefully check Appendix A, *Further reading*, especially the references listed in Section A.2.1, "General Linux resources", for relevant books and websites.

The appendix doesn't include references to books that focus on installation or system administration, so you'll need to find those on your own.

4. If you need to learn more about Linux GUIs, you may find Section 1.2, "Graphical user interfaces (GUIs) for Linux" and Appendix B, *Graphical alternatives* to be helpful as a starting point.

- If you're absolutely stuck and you're working with a Linux machine in front of you (that is, you're not connecting via SSH), start with the Linux GUI applications discussed in Appendix B, *Graphical alternatives* and then start again at the top of this section when you're ready to give the CLI another try.

Chapter 1. Getting started with Linux

This chapter provides some background information on user interfaces in Linux.

The most important information is the orientation to the command line in Section 1.3, “Accessing the Linux CLI through a GUI’s terminal window” and the sections that follow.

References to further reading on Linux can be found in Section A.2, “More on Linux”.

1.1. User interfaces in Linux: graphical and command line



Tip

If you’re using SSH,² you can skip to Section 1.4, “Shells, the shell prompt, and your home directory”.

Recall that graphical applications are unavailable while you’re using SSH, unless you’re using X tunneling.³

Fundamentally, there are two different ways to work with the Linux operating system:

- through a graphical user interface (GUI),¹ in which the user uses a mouse to manipulate windows.
- through the command line interface (CLI),² in which the user types commands at a prompt.

As suggested by the title, this guide focuses on the use of the CLI.

Discussion of GUIs is restricted to Section 1.2, “Graphical user interfaces (GUIs) for Linux”, Section 1.3, “Accessing the Linux CLI through a GUI’s terminal window”, and Appendix B, *Graphical alternatives*.

1.2. Graphical user interfaces (GUIs) for Linux³

If your primary use of Linux is not via an SSH connection,² your first experience with Linux will probably be with a graphical user interface called a *desktop environment*,⁴ which is often used in place of the lower-level command line-based approach.

³All of these GUIs use the X Window System [<http://www.x.org/>] (also called X) as their underlying windowing system. Wikipedia has an article [http://en.wikipedia.org/wiki/X_window_manager] that surveys the window managers that are compatible with X, as well as an article [http://en.wikipedia.org/wiki/X_Window_System] about X itself.

⁴Instead of a desktop environment, the GUI in use could be a window manager, such as Fluxbox [<http://fluxbox.org/>] or Enlightenment [<http://www.enlightenment.org/>]. A discussion of window managers is beyond the scope of this guide, although Wikipedia has an article [http://en.wikipedia.org/wiki/Window_manager] about them. Note that one component of a desktop

There are two particularly popular desktop environments for Linux:⁵

- GNOME [<http://www.gnome.org/>] (in which the menus are located at the top of the screen)
- KDE [<http://www.kde.org/>] or the K Desktop Environment (in which the K menu is located in the lower-left corner of the screen)

As explained in the next section, even if you're using a GUI, you can still access the CLI.

1.3. Accessing the Linux CLI through a GUI's terminal window

When you're using a GUI (in our case, a desktop environment), you can access the CLI through a *terminal window*, or terminal.

The procedure for bringing up a terminal varies depending on which desktop environment you are using.⁶

- If you're using GNOME, select the Applications menu from the top of the screen, then the Accessories submenu, then the Terminal application.

You can also press Alt-F2 and type `gnome-terminal` at the prompt.

- If you're using KDE, select the K menu from the lower-left corner of the screen, then the System menu, then the Konsole application.

You can also press Alt-F2 and type `konsole` at the prompt.

- If you're using a desktop environment other than GNOME or KDE, you'll need to consult the documentation for that environment.

1.4. Shells, the shell prompt, and your home directory

Once you open a terminal (or connect to a server via secure shell [SSH]),² you type in commands to interact with a program known as a *shell*.⁷ Although GNU Bash (the Bourne Again Shell) is the most popular shell for Linux, some Linux servers and the computers in some Linux computer labs use `tcsh` as the default shell.⁸ If you're interested in trying the Bash shell, see Section A.3.2, “Changing your shell temporarily to Bash”.

environment is a window manager, as Wikipedia's article [http://en.wikipedia.org/wiki/Desktop_environment] on desktop environments explains.

⁵While the Xfce [<http://www.xfce.org/>] desktop environment is also commonly used, it's not as widespread as GNOME and KDE, so this guide doesn't cover it.

⁶Desktop environments, including GNOME and KDE, are discussed in Section 1.2, “Graphical user interfaces (GUIs) for Linux”.

⁷The shell is covered in depth in Chapter 3, *Basic shell commands and related utilities*.

⁸For learning more about `tcsh`, Wikipedia's article [<http://en.wikipedia.org/wiki/Tcsh>] might be informative. You can also learn more by typing `man tcsh`, using the manual (`man`) pages that you'll learn about in Section 2.1, “Manual (“man”) pages”.

Once the shell starts up, you'll see the *shell prompt*, which indicates that the shell is ready for you to type a command.

Here is the prompt that I see when I log in to the CEC [<http://www.cec.wustl.edu/>]'s grid server: [jg18@grid ~]\$

Thus, the format being used is: [username@host_machine current_directory]\$

In addition, your current directory will initially be your *home directory*,⁹ as is indicated by the tilde (~) in the prompt above.¹⁰

1.5. Key differences between Windows/Mac OS X and Linux

Before you start working with the Linux CLI, you should be aware of some important differences between Linux and operating systems with which you're more familiar.

Unlike Windows or Mac OS X, Linux has a case-sensitive file system. This means that home, Home, and HOME would all be names for different directories. Similarly, as you will see in Section 3.2, “Navigating the file system”, you type **cd** to change directories: typing **CD** or **Cd** instead will not work.

When you're working with the CLI, you'll soon discover what Eric Raymond calls the “Rule of Silence: When a program has nothing surprising to say, it should say nothing.”¹¹ When a command finishes running (say, if you create a directory `foo` using **mkdir**), *there will be no confirmation message* of something like

```
directory `foo` created
```

Rather, there will only be a message if there's a problem, as in

```
mkdir: cannot create directory `foo`: File exists
```

Although you might use spaces in file and directory names in Windows or Mac OS X (such as `Paper due tomorrow.doc`), **don't use spaces in file or directory names in Linux**. Use underscores (the `_` character, usually on the same key as `-`) in a file or directory name (such as `Paper_due_tomorrow.doc`) instead of spaces. In fact, it's best if the only characters that you use in file and directory names are letters, numbers, hyphens (`-`), underscores (`_`), and periods (`.`), although you shouldn't start a file or directory name with a period, since, as noted in Section 3.2, “Navigating the file system”, doing so makes the file or directory hidden.

⁹Like the `C:\Documents and Settings\Your Username` directory in Windows or the `/Users/Your_Username` directory in Mac OS X, your home directory in Linux stores your personal files and settings. If you're a student, your home directory is probably the top-level directory for the long-term disk space (that is, not your “roaming profile”) that your school allocates for your use.

¹⁰The symbol `~` is listed in Table 3.2, “Common symbols from the file system”.

¹¹ From Eric Raymond's *The Art of Unix Programming* (Boston: Addison-Wesley, 2003). Found online at <http://www.catb.org/~esr/writings/taoup/html/ch01s06.html#id2878450>.

When you use a command that would make some permanent change in the file system (such as deleting files or renaming a file such that it would replace a pre-existing file), the system will not ask you

Are you sure?

You will only be prompted for confirmation if you explicitly ask for it, such as by using the `-i` command option that is mentioned in Table 3.3, “Commands for manipulating the file system”.

Similarly, the Linux CLI has no Trash or Recycle Bin for files that you decide to delete but might want to restore later. Deleted files are simply deleted.

1.6. Understanding the tables of commands and options

Throughout this guide are tables summarizing many of the most commonly used Linux commands and command options. Most table entries, like the one in the example below, follow a format similar to that used by Linux's built-in manual pages.¹² Some table entries, such as those found in Section 5.2, “GNU Emacs”, use a different format, although the differences are explained where relevant.

Since most commands have many options available, only a few options will be listed. To view all options for a given command, check the command's manual page.¹² Square brackets (that is, `[]`) around an item indicate that it's an optional argument for the command.

Example 1.1. How to read a table entry

Given the command synopsis

```
ls [-a] [-l] [path]
```

`ls` is the command, `-a` and `-l` are some of the options available for `ls`, and `path` is an optional argument for `ls`.

¹²Manual pages are discussed in Section 2.1, “Manual (“man”) pages”.

Chapter 2. Built-in help system

Before you try searching the Web for information on a command, consider checking Linux's extensive built-in help system first. In addition to the information in Section 2.1, “Manual (“man”) pages” and Section 2.2, “GNU info pages”, many (but not all) commands will provide basic usage information if they are invoked with the `--help` option. As an example, try typing `man --help` at the prompt.

Table 2.1. Commands for the built-in help system

| Command | Action |
|---|---|
| <code>man [section] name</code> | Manual pages |
| <code>info [menu-item]</code> | GNU info pages |
| <code>--help</code> option (on many commands) | Provides usage (how-to-use) information |

2.1. Manual (“man”) pages¹

You can start learning about the man pages by typing `man intro` or `man man` at the shell prompt.² The man pages’ help (which you can find by typing `h` while reading a man page) will give you the full list of navigation commands, but to start, you can scroll along a man page using `b` for back, `f` for forward, `<` to jump to the start of the man page, and `>` to jump to the end, typing `q` to quit. It is always worth checking the man pages when you are faced with an unfamiliar command or when you want to learn more about a command that you already know.

As mentioned in Section A.2.1, “General Linux resources”, a list of Web-based repositories of Linux manual pages can be found in the Wikipedia article [[http://en.wikipedia.org/wiki/Manual_page_\(Unix\)](http://en.wikipedia.org/wiki/Manual_page_(Unix))] on Unix manual pages.

2.2. GNU info pages

Info pages, which are another component of the built-in help system, are available for many software products from the GNU Project.³ To get started, try typing `info` or `info info` or `info some_program` (such as `info gcc`) at the prompt. The navigation commands are different from those for the man pages: `?` provides a list of commands and `h` starts a tutorial (but `q` still quits). It is worth checking the info pages for information on specific GNU programs.⁴ You can also find the built-in documentation for the GNU C library (or `glibc`) by typing `info libc` at the prompt.⁵

¹A historical note: as explained in an article [[http://en.wikipedia.org/wiki/Manual_page_\(Unix\)](http://en.wikipedia.org/wiki/Manual_page_(Unix))] from Wikipedia, the idea of built-in manual pages dates back to the original Unix operating system from Bell Labs.

²The shell prompt is discussed in Section 1.4, “Shells, the shell prompt, and your home directory”.

³The GNU (short for GNU’s Not Unix) Project “was launched in 1984 to develop a complete Unix-like operating system which is free software,” as explained on its website [<http://www.gnu.org/>].

⁴Even if you’re not sure whether `some_program` is a GNU product, try typing `info some_program` and see what happens.

⁵As explained on its official website [<http://www.gnu.org/software/libc/>], `glibc` is the GNU Project’s implementation of the C standard library, which is discussed in this Wikipedia article [http://en.wikipedia.org/wiki/C_standard_library].

Chapter 3. Basic shell commands and related utilities

Described by Joe Barr as a "web browser to the kernel,"¹ the shell is always running in a terminal, no matter what program may be in the terminal's foreground at any given time. As mentioned in Section 1.4, "Shells, the shell prompt, and your home directory", the shell in use is typically either **bash** or **tcsh**. You can determine the default shell for your system by typing **echo \$SHELL** at the prompt.

This chapter covers many of the critical commands for working with Linux, including those for using the file system and handling compressed files. The last two sections, Section 3.5, "Working with text streams" and Section 3.6, "Using job control (partial coverage)", are less crucial than the others, but they may prove useful as you become more experienced with using the shell.

Many of the commands discussed in this chapter belong to the GNU Core Utilities (or `coreutils`), a collection of basic utilities from the GNU Project.³ You can learn more about `coreutils` by typing **info coreutils** or from the resources listed in Section A.3.5, "The GNU Core Utilities (`coreutils`)".

References to further reading about the shell can be found in Section A.3, "More on the shell and related utilities".

3.1. Features of the shell

There are several useful features of the shell:

- You can access your command history (previously typed commands) using the up and down arrows.
- By pressing the Tab key, you can make the shell auto-complete command names and file/directory names *until it encounters ambiguity*.

For example, assuming that you have two text files, `foobar` and `foobaz`, in some directory with no other files in it, if you type **emacs f** at the prompt and then press Tab *without pressing Enter*, you will see the name partially completed to

```
emacs fooba
```

on the screen. The shell can't complete any further, however, since the last character could be `r` or `z`.

- If you type **!*some_string*** at the prompt and press Enter, the shell will check your command history for the most recently typed command beginning with `some_string` and

¹Barr, *CLI for Noobies*, 12.

If you aren't familiar with the concept of an operating system kernel, try the Wikipedia article [[http://en.wikipedia.org/wiki/Kernel_\(computer_science\)](http://en.wikipedia.org/wiki/Kernel_(computer_science))].

then automatically run that command. This can be useful if you typed a long command a while back and would rather not retype it or look through the command history to find it.²

- If you mistype or omit something as you type a command, you can edit what you've typed (assuming that you haven't pressed Enter yet) without having to retype everything.

The commands for command line editing are similar to those listed in Table 5.2, “Moving the cursor in Emacs” (with the exceptions that **M-<**, **M->**, **M-v**, and **C-v** are unavailable) and Table 5.3, “Manipulating text in Emacs” (but only the commands under “Deleting Text” are available), which are for the text editor GNU Emacs.³ You can also edit what you've typed with the left and right arrow keys and the Backspace and Delete keys.

To type a command like **C-a**, hold the Ctrl/Control key while typing **a**.⁴

To type a command like **M-f**, hold the Alt (or the Esc) key while typing **f**.⁴

In addition to the features listed above, GNU Bash (the Bourne-Again Shell) has other features⁵ that you can learn about from the resources listed in Section A.3.1, “General information on the Bash shell” and the sections that follow.

3.2. Navigating the file system

You can find your way around the Linux file system using the commands listed in Table 3.1, “Commands for navigating the file system” and the symbols listed in Table 3.2, “Common symbols from the file system”. Note that **ls** will not display hidden files or directories (whose names start with a period, such as `.history`) unless you include the `-a` option by typing **ls -a** at the prompt.

The command **find** is particularly powerful, although I think that the example below illustrates the most common use you'll have for it. To learn about all that **find** can do, consult **man find**.

Example 3.1. Using `cd`

cd (which is equivalent to **cd ~**) will take you to your home directory.⁹

cd foo will change your current directory to the subdirectory `foo`, displaying an error if `foo` doesn't exist.

⁵For example, if you're using Bash, try typing a single letter or a short string of letters (such as **fi**) and then pressing Tab twice.

Example 3.2. Using du

The command `du -ch ~` will tell you how much disk space each directory in your home directory⁹ takes up. Since this command can produce a lot of output to the screen, try viewing the result with `less`⁶ by typing `du -ch ~ | less` (the `|`, or “pipe,” is typically found on the same key used for the backslash (`\`)).

If you just want to see the *total* disk space usage only, type `du -ch ~ | tac | sed 1q` at the prompt (the next-to-last character is the number 1, not the letter l).⁷

Example 3.3. Using find

`find . -name "*.java"` will print the locations of all files whose names end with `.java` in the current directory and in all of its subdirectories.

Example 3.4. Using ls

`ls` (which is equivalent to `ls .`) will list files and directories in the current directory.

`ls -al ..` will list *all* files and directories (including hidden ones) in the directory that is one directory above `.` (the current directory) using a long-listing format.

Table 3.1. Commands for navigating the file system

| Command | Action |
|-----------------------------------|--|
| <code>cd [directory]</code> | Change to home directory [to specified directory] |
| <code>du -ch path</code> | Estimate file space usage, human-readable format with a grand total ^a |
| <code>file file_name</code> | Determine file type ^b |
| <code>find path expression</code> | Search for files in a directory hierarchy ^c |
| <code>ls [-a] [-l] [path]</code> | List contents of current directory [of the provided <code>path</code> instead] [include hidden files] [long listing format] ^d |
| <code>pwd</code> | Print name of current/working directory ^e |
| <code>which program_name</code> | Show the full path of a (shell) command ^f |

^aLinux manual page for `du`.

^bLinux manual page for `file`.

^cLinux manual page for `find`.

^dLinux manual page for `ls`.

^eLinux manual page for `pwd`.

^fLinux manual page for `which`.

⁶The command `less` is discussed in Section 3.5, “Working with text streams”.^a You can always check `man less` as well.

⁷For your enlightenment, `tac` is from the GNU Core Utilities (or `coreutils`; see Section A.3.5, “The GNU Core Utilities (coreutils)”), and `sed` (short for *stream editor*) filters text, as explained on GNU’s page [<http://www.gnu.org/software/sed/>] for its version of `sed`. In the example above, `sed` prints only the first line of the output. You can learn more about `tac` and `sed` by reading their man or info pages (such as `man tac` or `info sed`).

Table 3.2. Common symbols from the file system

| Symbol | Meaning |
|--------------------|---|
| ~ (tilde) | User's home directory ⁹ |
| . (period) | Current (working) directory |
| .. (double period) | One directory up from current directory |
| / (front slash) | Root (top-level) directory |
| * (asterisk) | Wildcard (can be used in many commands) |

3.3. Manipulating the file system

While being able to examine the file system is important, you'll also need to be able to make changes to it, by creating, copying, moving, and removing files and directories, as well as making symbolic links. Relevant commands can be found in Table 3.3, “Commands for manipulating the file system”. Note that **mv** can be used either to rename a file/directory or to move it, depending on the arguments passed to **mv**.



Caution

If **mv** is used to move a file to a place where a file with the name already exists or to rename a file such that a file with the new name already exists in that directory, the old file will be silently replaced, unless you specify the **-i** option when using **mv** (see Table 3.3, “Commands for manipulating the file system”).

Similarly, **cp** can silently replace files in the copying process. As with **mv**, you can specify the **-i** option to prevent **cp** from overwriting files.



Caution

Although you can use **rm** with the wildcard (*****) to quickly and easily remove files and directories, be careful: you can easily delete files that you did not mean to delete, and those deleted files cannot be recovered. This issue is particularly important when you are using the **-rf** option.

Example 3.5. Using **ln**

ln -s ~/classes/cse332/lab5 332lab5 will create a symbolic link (that is, an alias or shortcut) to `~/classes/cse332/lab5` called `332lab5`, placing the symbolic link in the current directory.

Example 3.6. Using mv

Assuming that `foo` is a directory and that `bar` is a file in `foo`:

`mv bar baz` will rename `bar` to `baz`, *silently replacing any pre-existing file in `foo` with the name `baz`.*

`mv bar ..` will move `bar` to one directory above `foo`, *silently replacing any pre-existing file in that directory with the name `bar`.*

`mv bar ../baz` will move `bar` to one directory above `foo` **and** rename `bar` to `baz`, *silently replacing any pre-existing file in that directory with the name `baz`.*

Example 3.7. Using rm

`rm *.txt` will delete all files whose names end in `.txt` from the current directory only.

`rm * .txt`, however, will delete **ALL** files (but not directories) in the current directory and will then try to delete a file named `.txt`.

*Therefore, when using **rm**, double-check your typing before you press Enter!*

`rm -rf foo` will delete `foo` regardless of whether it is a file or directory; if `foo` is a directory, then all of `foo`'s contents (including subdirectories) will also be deleted.

Table 3.3. Commands for manipulating the file system

| Command | Action |
|--|---|
| <code>cp [-i] [-r] source destination</code> | Copy files and directories ^a [prompt before overwrite] [include all subdirectories and their contents] |
| <code>ln -s target link_name</code> | Make symbolic link to <code>target</code> |
| <code>mkdir directory</code> | Create directory |
| <code>mv [-i] old_name new_name</code> | Rename file or directory [prompt before overwrite] ^b |
| <code>mv [-i] source destination</code> | Move file or directory [prompt before overwrite] ^b |
| <code>rm [-rf] [-i] files</code> | Remove files or directories ^c [include all subdirectories and their contents, with no prompt for confirmation] [prompt before any removal] |
| <code>rmdir directory</code> | Remove <i>empty</i> directory |

^aLinux manual page for `cp`.

^bLinux manual page for `mv`.

^cLinux manual page for `rm`.

3.4. Handling compressed files (tarballs)⁸

The **tar** utility (**tar** is short for *tape archive*) allows you to compress files and directories into a single file called a *tarball*. Note that compressing, listing, and extracting all use the same command (**tar**), but the options used differ depending on which action you wish to perform.

Like with all commands, you can find more information by checking **man tar**, but since the version of **tar** used on Linux is a GNU product, you can also find information by typing **info tar**.

Table 3.4. Commands for handling compressed files (tarballs)^a

| Command | Action |
|--|---|
| <code>tar -czf foo.tar.gz file_list</code> | Creates a tarball called <code>foo.tar.gz</code> from the list of files and directories (separated by spaces) in <code>file_list</code> |
| <code>tar -tzf foo.tar.gz</code> | Lists the files and directories in <code>foo.tar.gz</code> (does not extract them) |
| <code>tar -xzf foo.tar.gz</code> | Extracts the contents of <code>foo.tar.gz</code> and places them in the current directory |

^aAlthough, when used with the options above, **tar** will be silent (that is, it won't provide any message to the user unless there's a problem, as mentioned in Section 1.5, "Key differences between Windows/Mac OS X and Linux"), you can use the `-v` option for verbose output, as in `tar -czvf` or `tar -xzvf`; when used as `tar -tzvf`, a "long listing" output like that in `ls -l` is used.

3.5. Working with text streams



Caution

As with the commands **mv** and **cp** from Section 3.3, "Manipulating the file system", if you direct the I/O redirection operator `>` to a pre-existing file, that file will be silently replaced (overwritten). Thus you must be careful when selecting the name of the file to which `>` will direct its output.

The Linux CLI depends on programs communicating with each other (and with the user) through text streams,⁹ including through the commands that you type at the keyboard (called the *standard input* stream, or `stdin`) and the results or errors displayed on the terminal screen (called the *standard output* and *standard error* streams, or `stdout` and `stderr`). As such, you may benefit from having some ability to work with text streams.

The command **grep** and its variants (such as **egrep**) are particularly powerful and complex commands, since they're designed to search for textual patterns called *regular expressions*.¹⁰

⁸Joe Barr, *CLI for Noobies: A Primer on the Linux Command Line* (Boston: Prentice Hall, 2008), 151-155.

⁹Eric Raymond calls this the "Rule of Composition: Design programs to be connected with other programs" in *The Art of Unix Programming* (Boston: Addison-Wesley, 2003). Found online at <http://www.catb.org/~esr/writings/taoup/html/ch01s06.html#id2877684>.

¹⁰Wikipedia's article [http://en.wikipedia.org/wiki/Regular_expression] on regular expressions may be helpful.

Thus, consulting **man grep** and **info grep**, possibly in addition to checking sources such as those mentioned in Section A.1, “Suggested resources for finding information” for information on **grep**, is highly recommended.

More information on text processing with the shell can be found in Section A.3.3, “Text processing with the shell”.

More information on I/O redirection can be found in Barr, *CLI for Noobies*, Chapter 5 (“Everything’s a File”) and Garrels, *Introduction to Linux*, Chapter 5 (“I/O redirection”) [<http://tille.garrels.be/training/tldp/ch05.html>].

Example 3.8. Using cat with pipes

cat *.c | grep printf | less will search all files in the current directory whose name ends with `.c` for lines containing the string `printf` and then display the search results on the screen in the `less`^a read-only text viewer.¹¹

Example 3.9. Using /dev/null

./some_verbose_program > /dev/null will run `some_verbose_program` but discard its output by redirecting it to `/dev/null` (the “bit bucket”) instead of `stdout` (the terminal screen). If `some_verbose_program` has output that is directed to `stderr`, that output will still be displayed on the terminal screen.

Example 3.10. Using diff

diff file1 file2 > differences.txt will find all textual differences between `file1` and `file2` and store the results in a file (that will be created if it does not exist; if it does exist, it will be silently replaced) titled `differences.txt` without sending any of the results to `stdout` (the screen).

Example 3.11. Using echo

echo \$SHELL will display the value of the `SHELL` environment variable (that is, the full path of the default shell) on `stdout` (the terminal screen).

Table 3.5. Standard streams

| Name | Description | Default Location |
|---------------------|-----------------|------------------|
| <code>stdin</code> | Standard input | Keyboard |
| <code>stdout</code> | Standard output | Terminal screen |
| <code>stderr</code> | Standard error | Terminal screen |

¹¹This type of search will not indicate which files the lines containing `printf` came from, though. One way to obtain that information is to use **find** (discussed in Section 3.2, “Navigating the file system”), by typing **find . -name "*.c" | xargs grep printf** at the prompt. Be aware that while **cat** searches for files in the current directory only, **find** searches for files in the current directory *and all of its subdirectories* (and their subdirectories and so on).

Table 3.6. I/O redirection operators

| Operator in Context | Action |
|--------------------------------------|---|
| <code>command1 command2</code> | Direct the output from <code>command1</code> to the input of <code>command2</code> |
| <code>command1 < file1</code> | Direct the contents of <code>file1</code> to the input of <code>command1</code> |
| <code>command2 > file2</code> | Direct the output from <code>command2</code> to be stored as <code>file2</code> |
| <code>command3 >> file3</code> | Direct the output from <code>command3</code> to be appended to the contents of <code>file3</code> |

Table 3.7. Commands for working with text streams

| Command | Action |
|---------------------------------------|--|
| <code>cat file_list</code> | Concatenates the contents of the given list of files and sends the results to <code>stdout</code> |
| <code>diff file1 file2</code> | Performs line-by-line comparison of <code>file1</code> and <code>file2</code> , reporting any differences |
| <code>echo some_text</code> | Takes <code>some_text</code> from <code>stdin</code> and displays it on <code>stdout</code> |
| <code>grep pattern [file_list]</code> | Searches <code>stdin</code> by default [or the files in <code>file_list</code> instead] for lines matching <code>pattern</code> , then prints the results to <code>stdout</code> |
| <code>less file</code> | Allows read-only viewing of <code>file</code> ^a |

^a`less` has a diverse set of keyboard shortcuts available that seems to be designed to cover a wide range of common conventions for keyboard shortcuts, including those for the man pages and the `vi` and Emacs text editors. The commands mentioned in Section 2.1, “Manual (“man”) pages” ought to be enough to get by, but you can get the full listing of commands by typing `h` while you are using `less` (and typing `q` still quits).

3.6. Using job control (partial coverage)

Since it's now easy to open multiple terminals in either KDE or GNOME, job control (that is, handling multiple programs running simultaneously while being able to switch between them) is not as important as it used to be, when one terminal was all that was available. Thus, I will only cover how to kill (that is, terminate) a process (that is, a running program).

More information on processes and job control can be found in Barr, *CLI for Noobies*, Chapter 10 (“background, foreground, suspend”), and Garrels, *Introduction to Linux*, Chapter 4 (“Processes”) [<http://tille.garrels.be/training/tldp/ch04.html>], particularly Section 1 (“Processes Inside Out”) [http://tille.garrels.be/training/tldp/ch04.html#sect_04_01].

¹²I could also use `grep` (discussed in Section 3.5, “Working with text streams”) here: If I know the name (or part of the name) of the process that I want to terminate (in this case, `vim`), I could find its PID quickly by typing `ps -u jg18 | grep vim`.

Example 3.12. Using `ps` and `kill`¹²

```
[jg18@grid ~]$ ps -u jg18
```

```
  PID TTY          TIME CMD
 21947 ?            00:00:00 sshd
 21948 pts/6        00:00:00 tcsh
 23482 pts/6        00:00:00 vim
 23484 pts/6        00:00:00 ps
```

```
[jg18@grid ~]$ kill -9 23482
```

```
[1]      Killed                  vim
```

Table 3.8. Commands for terminating a process

| Command | Action |
|---------------------------------|--|
| <code>ps [-u user]</code> | List running processes with their process IDs [only those from the specified user] |
| <code>kill -9 process_ID</code> | Kill (end) the specified process |

Chapter 4. SSH (secure) file transfer (SFTP)



Tip

If you want to transfer *just one file* (or a directory compressed into a tarball³), and you know the exact path of the source and destination, you can use **scp** (a secure version of **cp**¹) instead of SFTP.

You can upload a file (which we'll call `my_file`) from the current directory on your machine to your home directory⁹ on the server by typing:

```
scp ./my_file your_username@server_name:/  
path_to_your_home_directory/my_file
```

To download a file (say, a file called `my_file` from the path `~/classes/cse332`² on the server to the current directory on your machine), type:

```
scp your_username@server_name:/  
path_to_your_home_directory/classes/cse332/my_file ./  
my_file
```

Note that you must specify the file name twice: on the local end (your machine) and on the remote end (the server).

If you're not sure what the path to your home directory on the server is, try typing **pwd** (mentioned in Table 3.1, “Commands for navigating the file system”) when you're connected to the server via SSH² and in your home directory (typing **cd** to get to your home directory if necessary, as shown in Example 3.1, “Using cd”).

Be careful, though: you can easily overwrite files (without notification from **scp** that you're doing so, of course) if you specify the wrong path -- the author managed to do this with his home page!

You can use SFTP to transfer files to or from a Linux server.

Start SFTP by typing **sftp username@server_name** at the prompt, filling in the appropriate values for your username and the server's name. The commands that you'll most likely need are listed in Table 4.1, “SSH file transfer (SFTP) commands”.



Note

SFTP commands look a lot like those discussed in Section 3.2, “Navigating the file system” and Section 3.3, “Manipulating the file system”, but commands to be run on the *local* machine (the one in front of you) have an **l** prefix (such as **lls**), while commands to be run on the *remote* machine (the one you're connecting to) don't have an **l** prefix (such as **ls**).

As far as I can tell, command history and command completion (discussed in Section 3.1, “Features of the shell”) are not available in SFTP. In addition, you cannot transfer entire directories through SFTP; you must compress them first into a tarball³ and then transfer the tarball.

Example 4.1. Using SSH file transfer (SFTP)

```
sftp jg18@grid.cec.wustl.edu
```

[Note that not even asterisks will be displayed on the screen as you type your password.]

You'll then have a prompt like `sftp>` at which you can type commands.

³Tarballs are discussed in Section 3.4, “Handling compressed files (tarballs)”.

Table 4.1. SSH file transfer (SFTP) commands

| Category | Command | Action |
|----------------------------------|---|--|
| <i>Basic SFTP Commands</i> | --- | |
| ~ | help or ? | View list of SFTP commands |
| ~ | exit or quit | Exit SFTP |
| <i>Working with File Systems</i> | --- | |
| ~ | (l)cd dir_name | Change directory on remote (local) machine |
| ~ | (l)ls [path] | List remote (local) directory contents |
| ~ | (l)mkdir dir_name | Create remote (local) directory |
| ~ | (l)pwd | Print name of working remote (local) directory |
| ~ | rm file (there is no "lrm") | Remove <i>remote</i> file |
| ~ | rmdir dir_name (there is no "lrmkdir") | Remove <i>remote</i> empty directory |
| <i>Downloading and Uploading</i> | --- | |
| ~ | get remote_path [local_path] | Download from remote_path to working local directory [to local_path instead] |
| ~ | put local_path [remote_path] | Upload from local_path to working remote directory [to remote_path instead] |
| <i>Other Commands</i> | --- | |
| ~ | !some_command | Execute some_command on local machine |

Chapter 5. Text editors

To work on your programming assignments, you will need to use a text editor. There are two editors available through the Linux CLI that are suitable for programming: vi and GNU Emacs.

Further reading on text editors can be found in Section A.4, “More on text editors”.

5.1. vi



Note

Substantial revisions to this section, including new tables of commands, are in the works. Stay tuned.

Vi (pronounced *VEE-EYE*)¹ “was the first real screen-based editor for Unix systems.”² It is present on *all* Unix and Unix-like operating systems (including Linux). Another frequently used text editor, GNU Emacs (discussed in Section 5.2, “GNU Emacs”), is *usually* (but not always) installed as well. You can start vi by typing **vi** or **vi file_to_edit** (such as **vi lab0.c**) at the prompt.

Unfortunately, I am not familiar with vi, so I cannot provide much help with using it. If you are interested in using vi, try typing **:help**³ when you are in vi to access its built-in help. You can also start a tutorial by typing **vimtutor** at the shell prompt. The most valuable command in any text editor is quitting without saving any changes, which is **:q!**³ in vi.

References to further reading on vi can be found in Section A.4.1, “More on vi”.

5.2. GNU Emacs

Described as “a complete working environment,” GNU Emacs has all of the functionality that you could ever want from a text editor.⁴ It is also, in my opinion, considerably easier to use than vi. You may wish to try both vi and Emacs and see which one you prefer. You can start Emacs by typing **emacs** or **emacs file_to_edit** (such as **emacs lab0.c**) at the prompt.

Emacs makes frequent use of the Control (Ctrl) and Meta keys, but don’t bother looking for the Meta key on your keyboard: I don’t know of any modern keyboards that have one. Instead, the Escape (Esc) key functions as the Meta key. In most configurations of Emacs, the Alt key also works as a Meta key. To type a command such as **C-x C-c**, press Ctrl and X, let go of the Ctrl and X keys, then press Ctrl and C.

Instead of working directly with files, when you use Emacs, you work with temporary buffers that affect the file on disk only when you save the buffer. When you create a file with **C-x C-**

¹Arnold Robbins, Elbert Hannah, and Linda Lamb, *Learning the vi and Vim Editors*, 7th ed (Sebastopol: O’Reilly, 2008), Chapter 1.

²Matthias Kalle Dalheimer and Matt Welsh, *Running Linux*, 5th ed., Section 19.1: “Editing Files Using vi.”

³Note that there is a colon (:) in the command. It’s easy to overlook it.

⁴Debra Cameron et al., *Learning GNU Emacs*, 3rd ed. (Sebastopol: O’Reilly, 2003), ix.

f, that file isn't saved to disk (that is, *actually* created) until you make a change in the initially blank buffer and then save it with **C-x C-s**.

Any text that you cut, copy, or delete (using the commands in Table 5.3, “Manipulating text in Emacs” below) is moved to the *kill ring*, where Emacs stores such text until you paste it (or yank it, as it is also called) using **C-y**. Do not confuse the kill ring with the *clipboard* that exists in Windows or in Mac OS X: text in the kill ring can only be used within Emacs. Graphical versions of Emacs, however, provide mechanisms for working with the system clipboard via the pull-down menu.

The **C-u** shortcut allows you to repeat a command for a given number of times. For example, to scroll down to the 50th line of a buffer when the cursor is initially at the top of the buffer, type **C-u 50 C-n** (without pressing Enter).

If you accidentally type the start of some command in Emacs, you can use the **C-g** shortcut to discard the partially typed command. **C-g** can also be used to interrupt a running Emacs command.⁵

You can use **M-x** to type in longer commands. For example, to switch into C++ editing mode, type **M-x c++-mode** and then press Enter. As with the shell, however, you can use Tab completion by typing **M-x c+** and then pressing the Tab key. You can also use Tab completion with file names when you are trying to open files with **C-x C-f**.

References to further reading on Emacs can be found in Section A.4.2, “More on GNU Emacs”.

⁵From the Emacs built-in tutorial.

Table 5.1. Basic Emacs commands

| Category | Command | Action |
|-------------------------------------|------------------|---|
| <i>Opening, Saving, and Exiting</i> | --- | |
| ~ | C-x C-f | Open (or create) file |
| ~ | C-x C-s | Save current buffer |
| ~ | C-x C-w | Save current buffer as... |
| ~ | C-x C-c | Exit Emacs |
| <i>Getting Help</i> | --- | |
| ~ | C-h ? | Emacs help |
| ~ | C-h t | Emacs tutorial |
| <i>Keys Found in Other Editors</i> | --- | |
| ~ | Arrow keys | Move cursor by one character/line |
| ~ | Delete/Backspace | Delete one character at a time |
| <i>Commands on Commands</i> | --- | |
| ~ | C-g | Quit command being typed (may need to press repeatedly) |
| ~ | C-_ | Undo previously typed command |
| ~ | C-u [number] | Repeat command to follow (default value is 4) |

Table 5.2. Moving the cursor in Emacs

| Category | Command | Action |
|--|---------|--------------------------------|
| <i>Moving to Start and to End</i> | --- | |
| ~ | M-< | Move to start of buffer |
| ~ | M-> | Move to end of buffer |
| ~ | C-a | Move to start of line |
| ~ | C-e | Move to end of line |
| <i>Moving Up and Down (Across Lines)</i> | --- | |
| ~ | M-v | Move up by one screen |
| ~ | C-v | Move down by one screen |
| ~ | C-p | Move to previous line |
| ~ | C-n | Move to next line |
| <i>Moving Forward and Backward (Within a Line)</i> | --- | |
| ~ | M-b | Move backward by one word |
| ~ | M-f | Move forward by one word |
| ~ | C-b | Move backward by one character |
| ~ | C-f | Move forward by one character |

Table 5.3. Manipulating text in Emacs

| Category | Command | Action |
|--|------------|---|
| <i>Deleting Text</i> | --- | |
| ~ | C-k | Delete from cursor to end of line |
| ~ | M-d | Delete from cursor to end of current word |
| ~ | C-d | Delete character |
| <i>Selecting, Cutting, and Pasting</i> | --- | |
| ~ | C-x h | Select all |
| ~ | C-spacebar | Set mark (for text selection) |
| ~ | C-w | Cut selected text (within Emacs) |
| ~ | M-w | Copy selected text (within Emacs) |
| ~ | C-y | Paste text |

Table 5.4. Additional Emacs commands

| Category | Command | Action |
|----------------------------|------------|---------------------------|
| <i>Search and Replace</i> | --- | |
| ~ | C-s | Incremental search |
| ~ | M-% | Find and replace |
| <i>Windows and Buffers</i> | --- | |
| ~ | C-x 2 | Split screen in two |
| ~ | C-x 1 | Recombine screen into one |
| ~ | C-x b | Switch to another buffer |
| ~ | C-x C-b | Bring up buffer list |
| ~ | C-x k | Kill (close) buffer |
| <i>Just For Fun</i> | --- | |
| ~ | M-x tetris | Play Tetris |
| ~ | M-x doctor | Emacs psychotherapist |

Chapter 6. Programming tools



Tip

Although the examples with this chapter are for programs written in C or C++, much of the information provided is still valid for programs written in other languages. If you want to be certain, though, you should check GCC and GDB's documentation, references to which are provided in Section A.5, "More on programming on Linux machines".

To compile and debug your programs, you'll need to use the programming tools that come with Linux, which are collectively known as the *GNU toolchain*. In this chapter, I'll describe some parts of the most commonly used components of the GNU toolchain, as well as how to run your compiled program in Linux.

References to further reading on programming in Linux and the GNU toolchain can be found in Section A.5, "More on programming on Linux machines".

6.1. Compiling programs

Programmers using Linux compile programs with the GNU Compiler Collection (GCC), a "compiler driver that invokes the language preprocessor, compiler, assembler, and linker, as needed on behalf of the user."¹ A list of commonly used compiler flags (which are options to adjust the compiler's behavior) is provided in Table 6.1, "Commonly used GCC flags". For a listing of all of the available flags, see `man gcc` or `info gcc` under the section "Invoking GCC."

To compile C source code or assembly programs with GCC, type: `gcc [flags] file_list`.



Note

When you are compiling C++ source code, type `g++` in place of `gcc`.

If you don't specify a file name for the resulting executable file using the `-o` compiler flag, GCC will use the default name of `a.out`.

Although you can use GCC directly to compile your programs, programmers typically use other software, such as GNU Make, for automating the build process. A discussion of Make is beyond the scope of this guide, but references to more information on Make are provided in Section A.5.3, "GNU Make and Makefiles".

References to further reading on GCC can be found in Section A.5.2, "More on GCC".

¹Randal Bryant and David O'Hallaron, *Computer Systems: A Programmer's Perspective* (Upper Saddle River: Prentice Hall, 2003), 541.

Example 6.1. Using GCC

`gcc -Wall -o foo foo.c` will attempt to compile `foo.c` to an executable, turning on (nearly) all compiler warnings, producing the executable `foo` if it is successful.

Table 6.1. Commonly used GCC flags

| Compiler Flag | Action |
|----------------------|---|
| <code>-c</code> | Compile to object code and then stop |
| <code>-g</code> | Include debugging information |
| <code>-ggdb</code> | Include extra debugging information for GDB ^a |
| <code>-I dir</code> | Check first for header files in directory <code>dir</code> ^b |
| <code>-L dir</code> | Check first for libraries in directory <code>dir</code> ^c |
| <code>-o name</code> | Set the (file) name of the compiler's output |
| <code>-O2</code> | Use the optimizer at level 2 |
| <code>-S</code> | Compile to assembly code and then stop |
| <code>-Wall</code> | Turn on all compiler warnings ^d |

^aGDB is discussed in Section 6.3, “Debugging programs”.

^b(can use this flag repeatedly to specify multiple directories)

^c(can use this flag repeatedly to specify multiple directories)

^d

Well, `-Wall` will actually turn on *most* compiler warnings. If, for some reason, you *really* want to know which warnings are enabled by `-Wall`, you can consult `man gcc`.

6.2. Running programs

If you try to run your compiled program (which we’ll call `lab0`) by typing `lab0` at the prompt, you’ll receive this message:

```
lab0: Command not found.
```

When you type a command at the prompt, the shell checks all directories listed in the `PATH` environment variable (which you can view by typing `echo $PATH`) to see if it is present there. If the shell can’t find it, it will print the message listed above.

By default, the current directory (`.`) is not included in the `PATH`, and although you can add it,² it is much easier (and strongly recommended) to simply type `./lab0` at the prompt instead,³ specifying that the shell should check the current directory for `lab0`.

²But see “Shell Startup File Elements” under “User Environments” in Brian Ward’s *How Linux Works* (San Francisco: No Starch Press, 2004) for why you shouldn’t add `.` to the `PATH`.

³Recall from Table 3.2, “Common symbols from the file system” that `.` (a period) stands for the current directory.

6.3. Debugging programs



Note

To use the GNU Debugger (GDB) effectively, you need to direct the compiler to include debugging information when it compiles your program. You can do this by using either the `-g` or the `-ggdb` compiler flag, as mentioned in Table 6.1, “Commonly used GCC flags”.

To debug a program using the GNU Debugger (GDB), type `gdb program_to_debug` (such as `gdb lab2`) at the prompt. You can then use GDB’s commands, some of which are listed in Table 6.2, “Commonly used GDB commands, Part 1” and Table 6.3, “Commonly used GDB commands, Part 2” below, to debug your program. GDB has many more commands than what is listed here; consult GDB’s built-in help for a complete listing. Also note that many commands have shortened versions: for example, you can type `r` in place of `run`. As with the shell (discussed in Section 3.1, “Features of the shell”), command history and command completion are available.

Finally, the command `x/` (listing the contents of memory) is a particularly powerful command, but it’s a fairly complex one, and describing it in detail is beyond the scope of this guide. Therefore, consulting `help x` while in GDB is recommended.

References to further reading on GDB can be found in Section A.5.4, “More on GDB”.

Table 6.2. Commonly used GDB commands, Part 1

| Category | Command | Action |
|---------------------------------|----------------|--|
| <i>Basic GDB Commands</i> | --- | |
| ~ | help (h) | GDB's built-in help |
| ~ | quit (q) | Quit GDB |
| <i>Controlling Program Flow</i> | --- | |
| ~ | continue (c) | Continue running the program (can use when a breakpoint is reached) |
| ~ | run (r) [args] | Run the program being debugged [with the supplied command line arguments (args)] |
| ~ | next (n) [#] | Step through program, over subroutines [repeat for specified number of times] |
| ~ | nexti (ni) [#] | Step through by one instruction [repeat for specified number of times] |
| ~ | step (s) [#] | Step through program, into subroutines [repeat for specified number of times] |
| ~ | stepi (si) [#] | Step by one instruction exactly [repeat for specified number of times] |
| ~ | kill (k) | Halt a program that's currently running in GDB |

Table 6.3. Commonly used GDB commands, Part 2

| Category | Command | Action |
|------------------------------------|--|---|
| <i>Breakpoints and Watchpoints</i> | --- | |
| ~ | <code>break (b) section</code> | Set a breakpoint at <code>section</code> |
| ~ | <code>delete [num_list]</code> | Delete all breakpoints [or just those specified, listed by breakpoint number] |
| ~ | <code>watch [expression]</code> | Set a watchpoint for <code>expression</code> |
| <i>Displaying Information</i> | --- | |
| ~ | <code>disas [function]</code> | Disassemble function in current frame [the given function instead] |
| ~ | <code>info (i) frame</code> | Display the state of the current stack frame |
| ~ | <code>info (i) registers</code> | Display the contents of the CPU registers |
| ~ | <code>list (l) [function]</code> | Show 10 lines of source code centered around the currently executing line [centered around the start of <code>function</code>] |
| ~ | <code>print (p) [&]variable</code> | Display the value of <code>variable</code> [the memory address of <code>variable</code>] |
| ~ | <code>x/16 address</code> | Display 16 words of data from memory starting at the provided address |

6.4. Inspecting and modifying programs

You can inspect and modify object files (including executables) using the GNU Binary Utilities (`binutils`), a set of tools that also includes the GNU assembler (`as`) and linker (`ld`).

Example 6.2. Using `objdump`

`objdump -d lab0 > lab0_dump.txt` will disassemble the executable file `lab0` and send the results to the file `lab0_dump.txt`, creating or overwriting the file as necessary.

References to further reading on `binutils` can be found in Section A.5.5, “More on the GNU Binary Utilities (`binutils`)”.

Table 6.4. Selected commands from GNU Binary Utilities (binutils)

| Command | Action |
|----------------------------------|---|
| <code>nm obj_file</code> | Display the symbol table of the given object file |
| <code>objdump -d obj_file</code> | Disassemble the given object file, sending the results to <code>stdout</code> |
| <code>strings obj_file</code> | List all strings found in the given object file |
| <code>strip obj_file</code> | Remove part or all of a given object file's symbol table |

Appendix A. Further reading

A.1. Suggested resources for finding information

For finding information quickly, searching with Google [<http://www.google.com/>] is often the best method. You can look up commands, symbols or messages that you don't recognize, and generic (that is, not specific to your program) compiler errors/warnings with Google. Wikipedia [http://en.wikipedia.org/wiki/Main_page] can also be a valuable resource, as can be seen by the numerous links to Wikipedia articles throughout this appendix and the guide as a whole. In addition, consider making the man and info pages (which are discussed in Chapter 2, *Built-in help system*) the first place you look for information about commands, given that they often have the information you need and are immediately available.

If you would prefer to read books, those published by O'Reilly [<http://oreilly.com/>] are often useful sources of information about Linux and other open source software products. Many of those books would be available through a public or university library, either on the shelf or electronically through Safari Books Online [<http://www.safaribooksonline.com/>] if your library is a subscriber. Selected relevant titles from O'Reilly and other publishers are listed in the sections that follow.

A.2. More on Linux

A.2.1. General Linux resources

The Linux manual pages (discussed in Section 2.1, "Manual ("man") pages") can also be found on the Web; the Wikipedia article [[http://en.wikipedia.org/wiki/Manual_page_\(Unix\)](http://en.wikipedia.org/wiki/Manual_page_(Unix))] on Unix manual pages has a list of repositories.

GNU has a list of manuals [<http://www.gnu.org/manual/>] for all of its software.

The Linux Documentation Project [<http://www.tldp.org/>] (TLDP) is a massive source of information, with many guides, HOWTOs, and other documents.

Prof. Norman Matloff at UC-Davis has a Unix and Linux Tutorial Center [<http://heather.cs.ucdavis.edu/~matloff/unix.html>], with numerous articles on Unix/Linux and programming in C.

LinuxCommand.org [<http://linuxcommand.org/>] has a short tutorial and some information on shell scripting and other topics.

Machtelt Garrels [<http://tille.garrels.be/>] has written an Introduction to Linux [<http://tille.garrels.be/training/tldp/>].¹

¹The direct link provided is for the "chunked" HTML version of that guide. If you want to pick a different format, check the TLDP's guides section [<http://www.tldp.org/guides.html>].

Google offers a Linux-only special search [<http://www.google.com/linux>].

As for books:

Barr, Joe. *CLI for Noobies: A Primer on the Linux Command Line*. Boston: Prentice Hall, 2008.

Barrett, Daniel J. *Linux Pocket Guide*. Sebastopol: O'Reilly Media, 2004.

Bovet, Daniel, and Marco Cesati. *Understanding the Linux Kernel*. 3rd ed. Sebastopol: O'Reilly Media, 2005.

Dalheimer, Matthias Kalle, and Matt Welsh. *Running Linux*. 5th ed. Sebastopol: O'Reilly, 2006.

Raymond, Eric Steven. *The Art of Unix Programming*. Boston: Addison-Wesley, 2003. This book can also be found online [<http://www.catb.org/~esr/writings/taoup/>] at the author's website.

Siever, Ellen, et al. *Linux in a Nutshell*. 5th ed. Sebastopol: O'Reilly, 2005.

Sobell, Mark G. *A Practical Guide to Linux Commands, Editors, and Shell Programming*. Upper Saddle River: Prentice Hall, 2005.

Stutz, Michael. *The Linux Cookbook: Tips and Techniques for Everyday Use*. 2nd ed. San Francisco: No Starch Press, 2004.

A.2.2. Linux file system

Binh Nguyen has a TLDP guide on the Linux Filesystem Hierarchy [<http://tldp.org/LDP/Linux-Filesystem-Hierarchy/html/index.html>].¹

You can also learn quite a bit about the file system from `man hier`.²

A.3. More on the shell and related utilities



Tip

You may also want to check the resources listed in Section A.2.1, “General Linux resources” for information on the Linux CLI.

A.3.1. General information on the Bash shell

GNU Bash has an online manual [<http://www.gnu.org/software/bash/manual/>].

Machtelt Garrels (mentioned in Section A.2.1, “General Linux resources”) has a Bash Guide for Beginners [<http://www.tldp.org/LDP/Bash-Beginners-Guide/html/index.html>].¹

²Barr, *CLI for Noobies*, 32.

Of course, you can also check **info bash**.

As for books:

Albing, Carl, and JP Vossen and Cameron Newham. *bash Cookbook: Solutions and Examples for bash Users*. Sebastopol: O'Reilly, 2007.

Kiddle, Oliver, and Jerry Peek and Peter Stephenson. *From Bash to Z Shell: Conquering the Command Line*. Berkeley: Apress, 2004.

Newham, Cameron. *Learning the bash Shell*. 3rd ed. Sebastopol: O'Reilly, 2005.

A.3.2. Changing your shell temporarily to Bash

If your default shell (which you can check by typing `echo $SHELL`) is not Bash and you'd like to try using Bash, here's one way to do so:

1. First, go to your home directory (in case you are not already there) by typing `cd` at the prompt.⁴
2. Next, check to see if the `.bashrc` file³ already exists by typing `ls .bashrc` at the prompt.⁴
3. If `.bashrc` does exist, then back it up by typing `mv .bashrc .bashrc.old` at the prompt,⁵ which will rename `.bashrc` to `.bashrc.old`.
4. Next, type the following two lines *exactly as they appear* (including all of the quotation marks but without any footnote references), pressing Enter after each line, to create your own `.bashrc` file:

```
echo 'export PS1="[bash \u@\h \W]$" ' > .bashrc678  
echo 'alias ls="ls --color"' >> .bashrc89
```

Now, to switch to Bash when you're using your default shell, just type **bash** at the prompt.

Compared to how my prompt looked in Section 1.4, “Shells, the shell prompt, and your home directory”, it now looks like this: `[bash jg18@grid ~]$`

To exit Bash and return to your default, type **exit** at the prompt. Note that if you want to exit the terminal when you're using Bash in this way, you'll need to type **exit** twice: once to exit Bash and then again to exit your default shell.

A.3.3. Text processing with the shell

The best article on text processing with the shell that I know of is "Unix for Poets", and since I'm not sure whether any given link to this article (in .pdf) will last, the best way to find it is through a Google search [<http://www.google.com/search?hl=en&q=%22unix+for+poets%22>].

A good companion to "Unix for Poets" is "egrep for Linguists" [http://stts.se/index.php?lang_id=en_uk&page=egrep].

As for books:

Dougherty, Dale, and Arnold Robbins. *sed & awk*. 2nd ed. Sebastopol: O'Reilly Media, 1997.

Friedl, Jeffrey. *Mastering Regular Expressions*. 3rd ed. Sebastopol: O'Reilly Media, 2006.

A.3.4. Shell scripting (using Bash)



Tip

The resources mentioned in Section A.3.1, “General information on the Bash shell” may also be helpful in learning about shell scripting.

Mendel Cooper [<http://personal.riverusers.com/~thegrendel/>] has written an Advanced Bash-Scripting Guide [<http://www.tldp.org/LDP/abs/html/index.html>].¹

As for books:

Robbins, Arnold, and Nelson H.F. Beebe. *Classic Shell Scripting*. Sebastopol: O'Reilly, 2005.

A.3.5. The GNU Core Utilities (`coreutils`)

GNU has an online manual [<http://www.gnu.org/software/coreutils/manual/>] for `coreutils`.

Wikipedia has an article [http://en.wikipedia.org/wiki/GNU_Core_Utilities] as well.

The blog Command Line Warriors [<http://commandline.org.uk/>] has a post entitled Ten Cool Coreutils Commands [<http://commandline.org.uk/command-line/2007/jan/6/ten-cool-coreutils-commands/>].

You can also find a great deal of built-in information by typing `info coreutils`.

A.4. More on text editors

A.4.1. More on vi

Vim (a popular extension of vi, short for *vi improved*) has its own website [<http://www.vim.org/>] and documentation [<http://vimdoc.sourceforge.net/>].

A ~575-page book [<ftp://ftp.vim.org/pub/vim/doc/book/vimbook-OPL.pdf>] on Vim is available on the Web (in .pdf), with errata for the book listed here [http://www.moolenaar.net/vim_errata.html].

Prof. Norm Matloff (mentioned in Section A.2.1, “General Linux resources”) has An Extremely Quick and Simple Introduction to the Vi Text Editor [<http://heather.cs.ucdavis.edu/~matloff/UnixAndC/Editors/ViIntro.html>].

Don't forget about Vim's built-in tutorial, which is best reached by typing `vimtutor` at the prompt in a terminal, although you can also get to it by typing `:help` while in vi/Vim.

As for books:

Robbins, Arnold. *vi Editor Pocket Reference*. Sebastopol: O'Reilly, 1998.

Robbins, Arnold, and Linda Lamb. *Learning the vi Editor*. 6th ed. Sebastopol: O'Reilly, 1998.

Robbins, Arnold, and Elbert Hannah and Linda Lamb. *Learning the vi and Vim Editors*. 7th ed. Sebastopol: O'Reilly, 2008.

A.4.2. More on GNU Emacs

GNU has a web page [<http://www.gnu.org/software/emacs/>] for Emacs, online Emacs manuals [<http://www.gnu.org/software/emacs/manual/index.html>], and An Introduction to Emacs Lisp Programming [<http://www.gnu.org/software/emacs/emacs-lisp-intro/emacs-lisp-intro.html>].

Prof. Norm Matloff (mentioned in Section A.2.1, “General Linux resources”) has a tutorial entitled Emacs: The Software Engineer's “Swiss Army Knife” [<http://heather.cs.ucdavis.edu/~matloff/UnixAndC/Editors/Emacs.html>].

Don't forget about Emacs's built-in tutorial, which you can find by typing `C-h t` while in Emacs.

There's also the Emacs info page at `info emacs`.

As for books:

Cameron, Debra. *GNU Emacs Pocket Reference*. Sebastopol: O'Reilly, 1998.

Cameron, Debra, et al. *Learning GNU Emacs*. 3rd ed. Sebastopol: O'Reilly, 2005.

Chassell, Robert J. *An Introduction to Programming in Emacs Lisp*. 2nd ed. Boston: GNU Press, 2004. This book can also be found online [<http://www.rattlesnake.com/intro/index.html>] at the author's website.

A.5. More on programming on Linux machines



Tip

As with all GNU software products, more information about the components of the GNU toolchain is available from their respective info pages (for example, by typing `info gcc` at the prompt).¹⁰

A.5.1. Linux programming and the GNU toolchain

Gareth Anderson's GNU/Linux Command-Line Tools Summary [<http://www.tldp.org/LDP/GNU-Linux-Tools-Summary/html/index.html>] is available as a TLDP guide [<http://www.tldp.org/guides.html>].¹

Peter Jay Salzman, Michael Burian, and Ori Pomerantz have written Linux Kernel Module Programming Guides for version 2.4 [<http://www.tldp.org/LDP/lkmpg/2.4/html/index.html>] and version 2.6 [<http://www.tldp.org/LDP/lkmpg/2.6/html/index.html>] of the Linux kernel.¹

Wikipedia has an article [http://en.wikipedia.org/wiki/GNU_toolchain] on the GNU toolchain.

As for books:

Bryant, Randal, and David O'Hallaron. *Computer Systems: A Programmer's Perspective*. Upper Saddle River: Prentice Hall, 2003.

Fusco, John. *The Linux Programmer's Toolbox*. Upper Saddle River: Prentice Hall, 2007.

Jones, M. Tim. *GNU/Linux Application Programming*. Hingham: Charles River Media, 2005.

Stevens, W. Richard, and Stephen A. Rago. *Advanced Programming in the UNIX Environment*. 2nd ed. Upper Saddle River: Addison-Wesley, 2005.

A.5.2. More on GCC

There's GCC's official homepage [<http://gcc.gnu.org/>], along with a set of the official manuals [<http://gcc.gnu.org/onlinedocs/>].

As for books:

Gough, Brian J., and Richard M. Stallman. *An Introduction to GCC*. Bristol: Network Theory Ltd., 2004.

von Hagen, William. *The Definitive Guide to GCC*. 2nd ed. Berkeley: Apress, 2006.

A.5.3. GNU Make and Makefiles

GNU Make has its own page [<http://www.gnu.org/software/make/>] and an accompanying manual [<http://www.gnu.org/software/make/manual/>].

As for books:

Mecklenburg, Robert. *Managing Projects with GNU Make*. 3rd ed. Sebastopol: O'Reilly Media, 2004.

A.5.4. More on GDB

GNU's website has documentation [<http://www.gnu.org/software/gdb/documentation/>] for GDB.

There's a "GDB Quick Reference" floating around the Internet that you can find with a Google search [<http://www.google.com/search?q=%22gdb+quick+reference%22>].

Peter Jay Salzman [<http://dirac.org/>] has a GDB Tutorial [<http://dirac.org/linux/gdb/>].

As for books:

[to be released in September 2008] Matloff, Norman, and Peter Jay Salzman. *The Art of Debugging with GDB, DDD, and Eclipse*. San Francisco: No Starch Press, 2008.

A.5.5. More on the GNU Binary Utilities (`binutils`)

The official page [<http://sourceware.org/binutils/>] for `binutils` is from sourceware.org [<http://sourceware.org/>].

Wikipedia has an article [http://en.wikipedia.org/wiki/GNU_Binutils] on `binutils` as well.

Appendix B. Graphical alternatives

Okay, I'll admit it: when I first took Linux-based programming courses in C and C++, I used graphical versions of some of these tools sometimes, and sometimes I even found myself more productive when I used the two together as needed. Therefore, I think that I should at least mention the existence of graphical alternatives to the tools I've presented.

As noted in Section 1.2, “Graphical user interfaces (GUIs) for Linux”, the two most popular desktop environments for Linux are GNOME and KDE. Thus, most of the software discussed in this appendix will be related to one of the two.

B.1. GUI-based file managers

A commonly used file manager for KDE is called Konqueror [<http://www.konqueror.org/>]; it can also be used as, among other things, a web browser.

In KDE 4, however, Konqueror has been replaced with Dolphin [<http://dolphin.kde.org/>] as the default file manager, although Konqueror can still be used as a file manager in KDE 4.

The file manager for GNOME is called Nautilus [<http://www.gnome.org/projects/nautilus/>].



Note

Be aware, though, that you will be able to work *much* more efficiently with the file system through the shell. Nonetheless, the file manager can be useful sometimes.

B.2. GUI-based file transfer programs

Options for graphical file transfer programs include gFTP [<http://gftp.seul.org/>] and KFTPgrabber [<http://www.kftp.org/>].

B.3. GUI-based text editors

B.3.1. GUI-based vi and Emacs

There are graphical versions of vi (such as gVim) and of Emacs in which pull-down menus and icons are available for use with the mouse. Be aware, however, that the more you use the keyboard instead of the mouse, the faster you can work, with the exception that selecting text for cutting, copying, or deleting may be faster with the mouse.

B.3.2. Other GUI-based text editors

The standard text editor for GNOME is gedit [<http://www.gnome.org/projects/gedit/>]; for KDE, there is Kate [<http://kate-editor.org/>] or KWrite [<http://kate-editor.org/kwrite>].

B.4. GUI-based programming and debugging tools

B.4.1. Integrated Development Environments (IDEs)

The Eclipse [<http://www.eclipse.org/>] IDE, which is particularly popular for Java development, which has an extension called the C/C++ Development Tools (CDT) [<http://www.eclipse.org/cdt/>].

KDE has an IDE called KDevelop [<http://www.kdevelop.org/>].

B.4.2. GUI-based debuggers not part of an IDE

One option for a GUI-based debugger is the Data Display Debugger (DDD) [<http://www.gnu.org/software/ddd/>].

A list of some other options can be found at Wikipedia's article [http://en.wikipedia.org/wiki/Debugger_front-end] on debugger front-ends.